This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

THIS PAGE BLANK (USPTO)

XP 000259620

2087 Elektronik
40(1991)6 August, No.16, Munchen, DE

Entwicklungssysteme

Emulatoren

E

p. 100 - 103 (h)

Dipl.-Ing. Stefan Richt, Peter Hamm

In-Circuit-Emulator wird „echtzeittauglich"

Praxis-Beispiel aus der Regelungstechnik

In zunehmendem Maße werden Mikrocontroller in Anwendungen eingesetzt, die sehr komplex und zeitkritisch sind und dazu noch ein hohes Maß an Zuverlässigkeit und Fehlertoleranz verlangen, so daß die Anforderungen an Entwicklungs- und Testgeräte – insbesondere an die In-Circuit-Emulatoren – immer höher geschraubt werden. Doch können die Emulatoren auch im Echtzeitbereich noch mithalten?

G11B15/46

Dieser Beitrag soll anhand eines Projektes die Eigenschaften beschreiben, die für Embedded-Controller-Entwicklungssysteme heute sehr wünschenswert sind. Beispielhaft herangezogen wird hierzu eine selbst konzipierte Entwicklungsumgebung.

Aufgabenstellung: Design eines Steuerrechners

Es sollte ein Steuerrechner mit einem 8032-Prozessor für ein analoges Kassettengerät entwickelt werden, der die Überwachung des Bandlaufes, die serielle Kommunikation und die Regelung der Bandgeschwindigkeit simultan für mehrere Kassettenlaufwerke übernehmen sollte. Als Programmiersprache war C vorgesehen.

Die Anforderungen an die Bandgeschwindigkeits-Regelung wurden sehr hoch gesteckt: Kurzzeitige Geschwindigkeitsabweichungen (Gleichlauf) durften maximal 0,07 % betragen, und die Langzeitkonstanz (Drift) sollte besser als 0,005 % sein. Die Bandlaufüberwachung mußte Fehlerzustände wie „Bandklemmen" und „Bandschlaufe" erkennen und verhindern, daß das Band beschädigt wird. Eine serielle Kommunikation mit einem externen Rechner sollte implementiert werden.

Die verwendeten Tools

Eingesetzt wurde ein selbstentwickelter In-Circuit-Emulator und ein ebenfalls selbstentwickelter Hardware-Simulator. In diesen Tools ließen sich Wünsche realisieren, die sich in mehreren Jahren bei der Entwicklung von „Embedded-Control"-Anwendungen herauskristallisiert hatten. In der Codierphase wurde hauptsächlich mit dem Simulator gearbeitet.

Die Implementierung der Firmware im Controller und das Testen bzw. Optimieren der Regelungs- und Überwachungsfunktionen erfolgte mit dem In-Circuit-Emulator.

Ohne „Interrupts" geht es nicht

Es zeigte sich, daß alle zeitkritischen Aufgaben in Interruptroutinen untergebracht werden mußten. Die Messung der Bandgeschwindigkeit wurde wegen der geforderten Meßgenauigkeit mit Hilfe der Capture-Register von Timer 2 vorgenommen. Die Kommunikation, die Überwachung des Bandlaufes sowie der Regelalgorithmus mußten als Interruptroutinen realisiert werden; lediglich der Interpreter, der die Kommandos der seriellen Schnittstelle bearbeitete, und die Ansteuerung der Stellglieder konnten im Hauptprogramm untergebracht werden. Die Regelung selbst wurde als Kaskadenregelung mit einer analogen Frequenzvorregelung realisiert, da der digitale Regelalgorithmus zu langsam war, um den geforderten Gleichlauf zu ermöglichen. Die Regelung wurde schließlich verschachtelt und die Zeitmeßeingänge und die Stellausgänge gemultiplext, um die Ansteuerung mehrerer Laufwerke zu ermöglichen.

Im Verlauf der Projekts stellte sich heraus, daß die Implementierung der Regelung der anspruchsvollste Teil der Aufgabe war.

Was macht das Testen einer digitalen Regelung schwierig?

Damit eine Regelung funktioniert, müssen mehrere Dinge gleichzeitig sichergestellt werden:

- O Der Regelalgorithmus muß fehlerfrei sein.
- O Der Fehler bei der Messung der Istgröße muß geringer sein als die zulässige Regelabweichung.

ENSDOCID: <XP___259620A__I_>

O Die Regelparameter müssen abgestimmt sein.
O Der Regelkreis muß geschlossen sein.

Module, die zur Regelung gehören, können zwar einzeln vorgetestet werden, dennoch läßt sich das dynamische Verhalten einer Regelung nur in Echtzeit beurteilen. Gerade hier sind leistungsfähige Entwicklungswerkzeuge gefragt, die über erweiterte „Echtzeit"-Testmöglichkeiten verfügen.

Die Interruptroutinen: aktiv im Hintergrund

Der Steuerrechner sollte neben der Regelung der Bandgeschwindigkeit auch die Kontrolle der seriellen Schnittstelle und die Überwachung des Bandlaufes erledigen. Der Bandlauf läßt sich aber erst dann sinnvoll überwachen, wenn die Bandgeschwindigkeit korrekt eingestellt ist: Deshalb mußten die Interruptroutinen der Regelung zuerst implementiert werden und während des gesamten Systemtests im Hintergrund aktiv bleiben.

Mit einem Emulator, der die Abarbeitung von Interruptroutinen bei stehender Emulation nicht unterstützt, hätte man diese Anwendung nur noch mit dem Trace (Verfolgunsspeicher) testen können, nachdem die erste Interruptroutine implementiert worden war. Dies wäre jedoch eine sehr unkomfortable Lösung.

Da der Mikrocontroller-Emulator auch die Abarbeitung von Interruptroutinen im Hintergrund unterstützen sollte, wurde eine Interrupt-Erkennung und Prioritäten-Zuordnung implementiert, die universell mit allen 8051-Derivaten zusammenarbeitet, bis zu vier Prioritätsebenen berücksichtigt und vom Anwender keinerlei Eingaben erfordert.

Dadurch ist es möglich, das Programm wie gewohnt zu „testen", d.h. alle zur Verfügung stehenden Debug-Befehle wie „Single-Step", „Snapshot" oder „Breakpoint" auszuführen, während im Hintergrund alle aktiven Interruptroutinen in Echtzeit abgearbeitet werden.

Eine ungewohnte Programmtest-Strategie

Daraus ergibt sich eine für viele Emulator-Besitzer ungewohnte, da von ihren Geräten nicht unterstützte, aber sehr effiziente und bequeme Testmethode: Man implementiert und prüft die Interruptroutine mit der höchsten Priorität zuerst. Ist diese „wichtigste" Interruptroutine getestet, kommen die Interruptroutinen niedrigerer Priorität an die Reihe, wobei die „wichtigste" Interruptroutine weiterhin in Echtzeit läuft. Sind alle Interruptroutinen getestet und fehlerfrei, kann zuletzt bequem das Hauptprogramm geprüft und optimiert werden, wobei alle Interruptroutinen weiterhin in Echtzeit abgearbeitet werden.

Interaktive Abstimmung der Regelparameter bei laufender Regelung

Während die Regelung in Echtzeit lief, konnten mit dem Emulator interaktiv die Parameter geändert werden. Da die Wirkungen unmittelbar sichtbar waren, ließ sich die Regelung bequem abstimmen. Es wurde ein PID-Regler eingesetzt; zunächst als reiner P- (Proportional-)Regler implementiert. Der Proportional-Anteil wurde anschließend so lange verändert, bis das System an die Stabilitätsgrenze kam. Die Einstellung der I- (Integral-), D- (Differential-) und P- Anteile erfolgte zunächst grob nach Einstellregeln und konnte interaktiv schnell am System optimiert werden.

Ohne einen leistungsfähigen Analysator geht es nicht

Es gibt Situationen, in denen ein Entwickler über einen leistungfähigen Trace-Analysator verfügen sollte, denn komplizierte Soft- oder Hardwarefehler lassen sich meist nur damit aufspüren.

Was heißt „unscharf triggern"?

Angenommen sei das Beispiel, in dem sich ein Fehler dadurch zeigt, daß eine Float-Variable einen falschen Wert annimmt. Hier ist nun nicht auf den Wert der Variablen selbst zu triggern, sondern es genügt, auf die Adresse der Variablen zu triggern, was mit jeder Breakpoint-Logik einfach möglich sein sollte. In den Trace-Speicher werden alle Speicherzugriffe auf die Variable inclusive Pre-Trace geschrieben. Der Trace-Speicher läßt sich nun mit oder ohne Software-Unterstützung analysieren und somit die fehlerhafte Variable suchen. In der zusätzlich zu der Variablen aufgezeichneten Vorgeschichte (Pre-Trace) kann die Ursache des Fehlers lokalisiert werden.

Die meisten Emulatoren verfügen als Zusatzoption über einen solchen Analysator. Abhängig von einem Triggersignal werden dabei Bus-Zyklen und Daten von Probes in einen Speicher geschrieben, dessen Inhalt man zur Programmanalyse wieder auslesen kann. Bekannte Aufzeichnungsarten sind Pre-, Post- und Center-Trace-Modus. Im Pre-Trace-Modus ist die Aufzeichnung zunächst aktiv, mit dem Triggersignal wird die Aufzeichnung gestoppt. Im Speicher stehen die Daten zum Zeitpunkt des Triggersignals (Triggerereignis) und Daten, die zeitlich vor dem Triggersignal lagen (Pre-Trace). Im Post-Trace-Modus wird die Aufzeichnung durch das Triggersignal gestartet und bei einer bestimmten Abbruchbedingung (z. B. nach 100 Zyklen) wieder gestoppt. Im Speicher stehen das Triggerereignis selbst und Daten, die nach dem Triggersignal aufgetreten sind (Post-Trace). Der Center-Trace-Modus stellt eine Kombination aus beiden Modi dar. Das Triggersignal wird von einer Breakpoint-Logik erzeugt, die der

DOCID: <XP___259620A__I_>

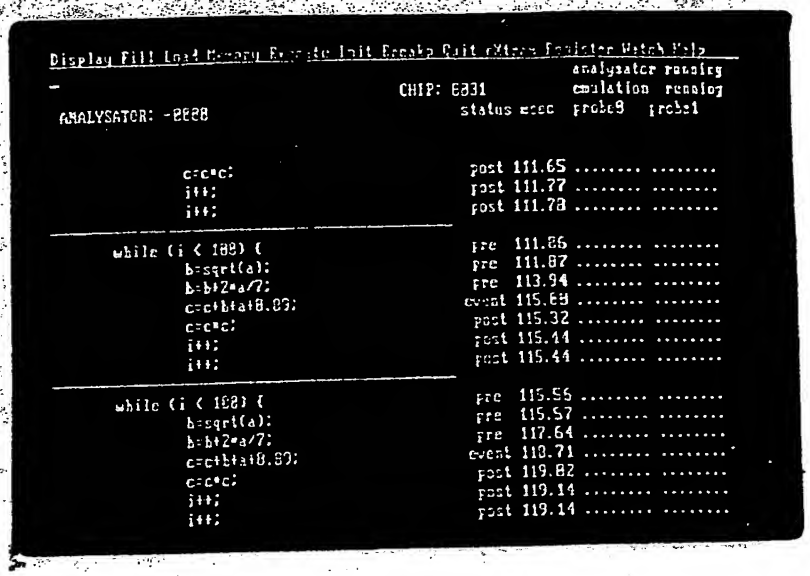

Bild 1. Multiple-Center-Trace-Aufzeichnung auf Source-Code-Ebene: aktives Source-Line-Filter, Pre-Trace mit zwei Zyklen, Post-Trace mit drei Zyklen

Entwickler programmieren kann, um die Aufzeichnung zu steuern.

Ist „Single Pre-Trace"-Modus ausreichend?

Da die Ursachen für Fehlerzustände zeitlich vor dem Auftreten des Fehlerzustandes selbst liegen, ist der Pre-Trace-Aufzeichnungsmodus meist das wirkungsvollste Mittel zur Fehlersuche. Nachteilig bei dem bekannten Pre-Trace-Modus ist die Tatsache, daß technisch bedingt nur ein Ereignis mit Pre-Trace im Speicher stehen kann.

Probleme treten dann auf, wenn auf die Auswirkung eines unbekannten Fehlers nicht getriggert werden kann. Bei der implementierten Regelung mußte man beispielsweise die Ursache für sporadisch auftretende, zu hohe Gleichlaufschwankungen finden. Wie kann man aber auf derartige „Ausreißer" triggern, ohne das Gleichlauf-Meßgerät umzubauen und mit einem Triggerausgang zu versehen? Ein anderes Beispiel kann ein Fehler sein, bei dem eine Float-Variable einen unerlaubten Wert annimmt.

Selbst mit einer äußerst leistungsfähigen sequentiellen Breakpoint-Logik kann in Echtzeit kaum auf den Wert einer Float-Variablen getriggert werden. – Doch es gibt einen Ausweg.

„Multiple Center-Trace" ist die Lösung

Mit Hilfe des „Multiple Center-Trace"-Modus ist es möglich, auch „nichttriggerbare Fehler" im Trace-Speicher aufzuzeichnen und somit zu finden. Doch was bedeutet „Multiple Center-Trace"-Modus?

Zusätzlich zu jedem Triggerereignis können ein Pre-Trace und ein Post-Trace aufgezeichnet werden. Die Länge von Pre- und Post-Trace ist getrennt einstellbar und kann zwischen 0 und 512 Zyklen liegen (*Bild 1*). Ein Speichereintrag, der aus Triggerereignis, Pre- und Post-Zyklen besteht, läßt sich dann treffend als „Frame" bezeichnen. In Echtzeit und ohne Unterbrechung der Trace-Aufzeichnung können beliebig viele „Frames" in den Trace-Speicher aufgezeichnet werden. Die Anzahl der „Frames" hängt von der eingestellten Länge des Pre- und des Post-Trace sowie von der Tiefe des Trace-Speichers ab. Im 32 KWorte tiefen Trace-Speicher des Emulators lassen sich z. B. 16 KFrames mit jeweils einem Pre-Trace-Zyklus aufzeichnen.

Durch den „Multiple Center Trace"- Modus ist man nun bei der Fehlersuche in der Lage, „unscharf" zu triggern, das heißt, es ist nicht notwendig, auf die exakte Fehlerbedingung selbst zu triggern, sondern es genügt, die Triggerbedingung so zu definieren (*Bild 2*), daß wenigstens einer der aufgezeichneten Frames den Fehler enthält.

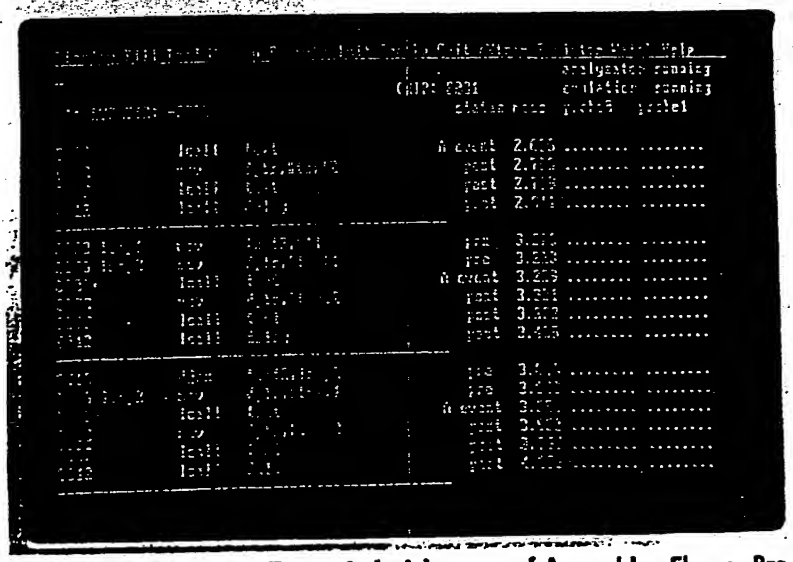

Bild 2. Multiple Center-Trace-Aufzeichnung auf Assembler-Ebene: Pre-Trace mit zwei Zyklen, Post-Trace mit zwei Zyklen

BNSDOCID: <XP____259620A__I_>

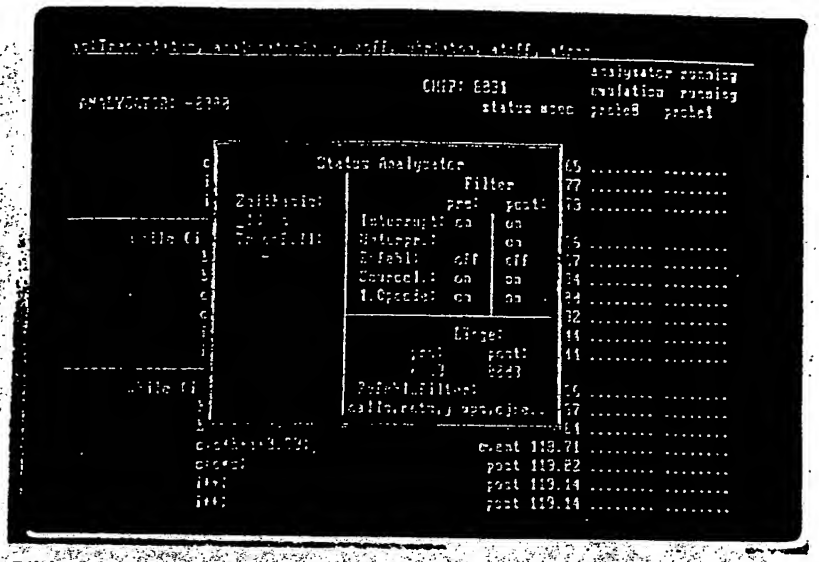

Bild 3. Details zum Analysatorstatus

Durch „Filter" den Trace-Speicher optimal nutzen

Gerade beim Hochsprachen-Trace kann es vorkommen, daß der Trace-Speicher schlecht genutzt wird. Es ist nämlich durchaus keine Seltenheit, wenn eine Hochsprachen-Zeile mehr als 1000 Assemblerbefehle enthält. In einer ungefilterten Aufzeichnung würden so nicht einmal 32 Quellcode-Zeilen in einen 32 KWorte tiefen Trace-Speicher passen!

Im hier beschriebenen Emulator wurden deshalb dem Trace-Speicher verschiedene Filter vorgeschaltet. Durch Setzen dieser Filter kann der Entwickler erreichen, daß nur Daten, die ihn interessieren, aufgezeichnet werden. Dies erhöht einerseits die Übersichtlichkeit der Aufzeichnung, andererseits spart dieses Verfahren Speicherplatz, so daß Langzeittests überhaupt erst möglich werden. Durch Ausnützen der Filter und der sehr leistungsfähigen sequentiellen Breakpoint-Logik kann die Aufzeichnungsdauer extrem verlängert werden (z.B. über Nacht). Dies ist hilfreich bei Langzeittests, wenn also ein Fehler nur selten auftaucht. Eine solche Situation wäre z.B. ein Stack-Überlauf nach Aufruf einer Interruptroutine, die ihrerseits eine sehr speicherintensive Hauptprogrammroutine unterbricht.

Folgende Filter, die voneinander unabhängig und für Pre- und Post-Trace aktiviert werden können, wurden implementiert:

- Unterprogramm-Filter: Aufgezeichnet wird nur der Unterprogrammaufruf; das Unterprogramm selbst jedoch nicht. Dadurch kann erreicht werden, daß ein bereits getestetes Unterprogramm nicht mehr aufgezeichnet wird. Dies erhöht die Lesbarkeit einer Trace-Aufzeichnung wesentlich.
- Interrupt-Filter: Das Filter verhindert, daß Interruptanforderungen die Trace-Aufzeichnung des zu testenden Programmteils unterbrechen. Dieses Filter funktioniert im wesentlichen genauso wie das Unterprogramm-Filter.
- Source-Line-Filter: Dieses Filter wurde speziell für Anwendungen in Hochsprache implementiert. Aufgezeichnet werden nur diejenigen Befehle, die einer Hochsprachenzeile entsprechen. Dies führt zu einer äußerst komprimierten und Trace-Speicher sparenden Aufzeichnung von C-Programmen. Selbstverständlich erfolgt die Anzeige des Trace-Speichers auch auf Source-Code-Ebene.
- Befehls-Filter: Nur bestimmte Gruppen von Befehlen (z.B. alle Sprungbefehle) werden aufgezeichnet. Dies ist z.B. für folgende Testsituation interessant: Ein Unterprogramm soll getestet werden, wobei wichtig ist, welche Routinen zuvor aufgerufen wurden. Dazu aktiviert man im Pre-Trace das Befehls-Filter und zeichnet nur alle Unterprogrammaufrufe (Befehl „call ..") auf. Im Post-Trace ist das Filter inaktiv; das komplette Unterprogramm wird aufgezeichnet.
- Zyklus-Filter: Die CPUs der Intel-8051-Familie führen pro Befehl bis zu acht Speicherzugriffe durch. Für einen Großteil aller Testanwendungen genügt jedoch der erste Speicherzugriff, da mit diesem der auszuführende Befehl festgelegt wird und die weiteren Opcodes (z.B. RAM-Adressen) später aus dem bekannten Programm rekonstruiert werden können. Ist dieses Filter aktiviert, so werden nur die ersten Speicherzugriffe und alle Zugriffe auf ein externes RAM aufgezeichnet.
- Frei programmierbares Filter: Spezielle Programmteile können mit einem Cursor markiert und somit zur Aufzeichnung freigegeben werden. Dieses Filter läßt sich sehr flexibel einsetzen (siehe auch *Bild 3*).

Stefan Richt studierte Elektrotechnik an der TU München und ist seit 1986 bei der Firma Richt-Magneton GmbH in der Entwicklung tätig. Nebenberuflich betreibt er ein Entwicklungslabor, in dem u.a. der vorgestellte In-Circuit-Emulator entwickelt wurde.

Peter Hamm studierte Physik an der TU München und arbeitete freiberuflich seit 1987 bei der Firma Richt-Magneton GmbH. Bei der Entwicklung des vorgestellten In-Circuit-Emulators war er maßgeblich beteiligt.

BNSDOCID: <XP____259620A__1_>